

LINUX EXPERT

FAQ

Q How would I go about creating an RPM from a source tarball?

A You can create your own RPMs easily if you have a tar.gz file that contains a program's installation files.

This has a few advantages over running the 'configure', 'make' and 'make install' commands, not least of which is that it will also take care of installing all the dependencies at the press of a button. For this, you need to use a graphical installation tool such as YaST or the Synaptic Package Manager included with Ubuntu and other Debian-based distributions.

Creating RPMs is done using the `rpmbuild` command. The actual command line is simple:

```
# rpmbuild -tb <tarball
name>.tar.gz
```

Because `rpmbuild` will build from the tarball, you'll need to have all the dependencies upon which the tarball relies installed too, as if you were building and installing it for real.

Once built, the RPM can then be installed on this or another computer using the command:

```
# rpm -i <package
name>.rpm
```

As it's open source, you're also free to distribute it yourself.

Jon Thompson
UNIX/Linux management
and security consultant



linux@computershopper.co.uk

Virtual private networks

A virtual private network enables you to encrypt traffic to keep it secure. Jon Thompson explains how to set up and run a VPN to keep your data safe

Communications between Linux servers and Windows desktops are often unencrypted, which is a problem if you need to access services privately. You might be away from home and want secure access to your own mail server, for instance. Or you might have a wireless network that carries personal, private or commercially sensitive data, and need to prevent people intercepting your traffic. You might simply need to secure communications between two adjacent offices. All these scenarios have one solution in common: encryption.

We covered the use of SSL to secure communications in *Linux Expert*, *Shopper* September 2005, but for many people encryption still appears to be difficult to set up and to manage. The perception is that you need to fiddle about with port numbers, certificates and other paraphernalia each time you want to connect. Our previous article dispelled this myth regarding individual ports, but what if you simply want to encrypt all your traffic? This is simple to achieve using a virtual private network (VPN). With a VPN, you can automatically create a secure tunnel through which all traffic flows. The VPN encrypts all data and sends it over a single port.

In this month's *Linux Expert*, we show you how to set up and run a VPN that's completely transparent once running, and easily configured to provide secure access to your servers. But before we show you how to set it up, we'll look at the technology behind it.

INTRODUCING OPEN VPN

To create our VPN, we'll be using the open-source OpenVPN by James Yonan. This is a mature and very configurable piece of software that runs on a large number of operating systems, including Linux, Windows 2000/XP, OpenBSD, FreeBSD, NetBSD, Mac OS X and even Sun's Solaris. This means we don't need to limit ourselves when choosing which operating systems to use for both servers and clients.

OpenVPN uses openSSL to secure data. Both the proprietary SSL and the open-source openSSL are designed to transport data securely. Created by Netscape, the SSL standard uses a two-key system. To encrypt data there's a public key that, though it can encode data, cannot be used to decode it again. Decoding is performed using a private key known only to the recipient of the message. Web addresses that begin with https use SSL or openSSL to encrypt all traffic between the browser and the server.

When you want to connect, the client first sends a ClientHello message to the server. This contains a list of the cipher and compression methods it understands and the highest level of the SSL protocol it can use. The client then receives back a ServerHello message indicating the choices the server has made from the options sent to it. The client and server swap their bona fides, called certificates. Depending on the configuration, these certificates are independently verifiable by both parties, because they can also be held by a trusted third-party organisation such as VeriSign.

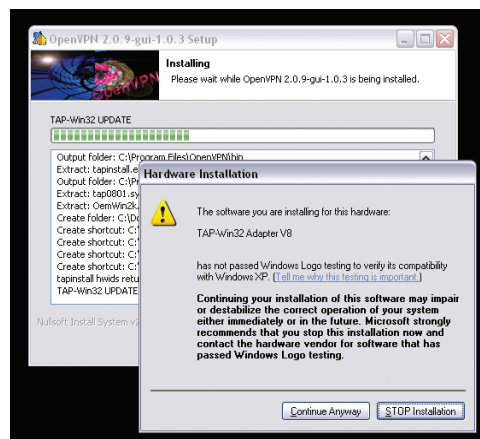
This third party is called the Certificate Authority (CA). In commercial operations, the client and server contact the CA and receive back each other's certificate, proving they are who they claim. Contained within this data are the public keys the client and server use to encrypt messages before sending them.

Once both parties have swapped certificates and public keys, and know what protocols and encryption methods they will use, OpenVPN creates a virtual interface over which it tunnels data between the two. This is called a TUN and it enables the VPN server to distinguish between multiple interfaces. This is useful in situations where you install a VPN on a computer with more than one network interface card. You might want to do this in situations where you're using a Linux computer to act as a secure firewall and router and decide to use it to host your VPN, too.

INSTALLATION

OpenVPN comes as standard with all mainstream Linux distributions. Because we'll be using some of the example files and utilities that come with the tarball used to distribute it, we're going to build the source distribution on the server. We recommend you install the RPM on the client, however, as this involves far less configuration work. You can either use the graphical installation tool that came with the distribution or install it by hand with the `rpm -i` command. The client and server parts of the VPN use the same software, which is handy.

To build the source tarball, first make sure you have a C compiler available. You then need to install a couple of libraries upon which the build depends. The first is the LZO library. Again, this should come with all mainstream Linux distributions, but if yours does not contain it you can download it from www.oberhumer.com/opensource/lzo/download. The latest tarball is `lzo-2.02.tar.gz`. Unpack and build it as usual with the following commands:



▲ Windows XP might make a fuss when you install the OpenVPN client or the GUI version, but click Continue Anyway to carry on

```
# tar xzvf lzo-2.02.tar.gz
# cd lzo-2.02
# ./configure
# make
# make install
```

The second library you need from your distribution disk is the openssl development package. The package is called openssl-devel. This is almost certain to be included in any modern Linux distribution, so either use your graphical installation tool, if you have one, or install the RPM as follows:

```
# rpm -i <openssl-devel package>
```

You can also download and install the source tarball from www.openssl.org/download. Unpack, configure and install it as follows:

```
# tar xzvf openssl-0.9.8e.tar.gz
# cd openssl-0.9.8e
# ./configure
# make
# make install
```

With a C compiler on hand, and the LZO and openssl libraries installed, we're in a position to unpack, configure and build OpenVPN as follows:

```
# tar xzvf openvpn-2.0.9.tar.gz
# cd openvpn-2.0.9
# ./configure
# make
# make install
```

We also need to create a link from /usr/sbin to the directory where we built the OpenVPN system so we can simply run the command as root without specifying the path. We can do this as follows:

```
# ln /usr/sbin/openvpn <path to
the openvpn directory we created
earlier>
```

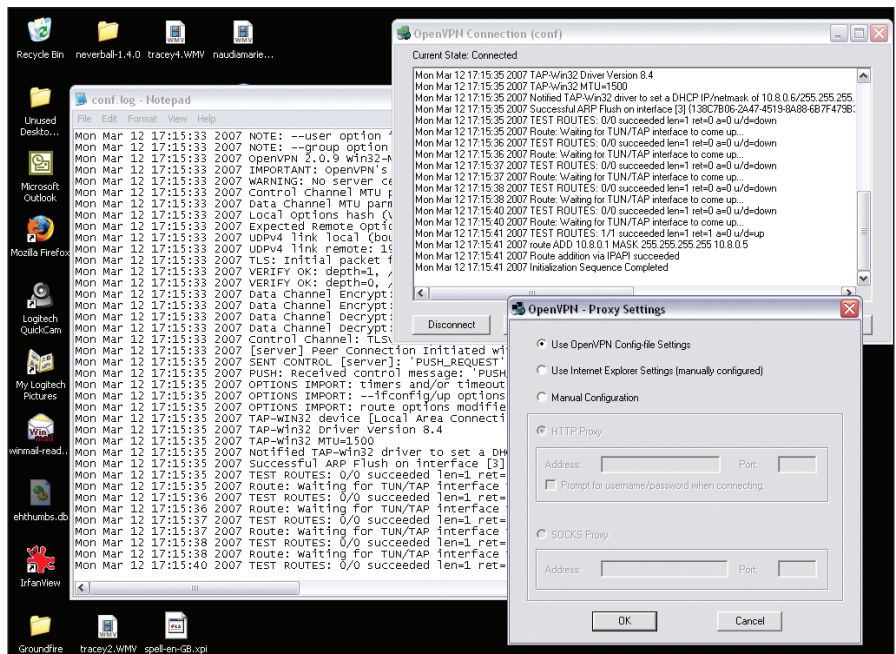
If your system needs to traverse a firewall, remember to open port 1194, which is the port used by OpenVPN to tunnel all its traffic. Do this before you start configuring and using OpenVPN.

TESTING TIMES

We're now in a position to test the installation and make sure that OpenVPN will encrypt and decrypt properly. Generate a key file by entering the following commands:

```
# openvpn --genkey --secret key
# openvpn --test-crypto --secret key
Thu Mar 8 13:34:59 2007 TESTING
ENCRYPT/DECRYPT of packet length=1
Thu Mar 8 13:34:59 2007 TESTING
ENCRYPT/DECRYPT of packet length=2
Thu Mar 8 13:34:59 2007 TESTING
ENCRYPT/DECRYPT of packet length=3
Thu Mar 8 13:34:59 2007 TESTING
ENCRYPT/DECRYPT of packet length=4
Thu Mar 8 13:34:59 2007 TESTING
ENCRYPT/DECRYPT of packet length=5
```

The output will scroll past for packet lengths up to 1,500 bytes. We can now test connectivity using a series of automated tests as follows. First, open two command lines on the server and, as root, type the following:



▲ With Windows you can start the VPN without resorting to the command line by using OpenVPN GUI

```
# openvpn --config sample-config-files/
loopback-client
```

The sample-config-files directory is found under the top-level OpenVPN directory. This command will complain about its connections being refused by OpenVPN's server portion until we enter the following in the second window to start the server:

```
# openvpn --config sample-config-files/
loopback-server
```

The tests will run for anything up to about two minutes, pausing for several seconds between outputting the results of each one.

POINT-TO-POINT CONFIGURATION

We can then create a basic, point-to-point VPN that will use just one client. This is all many people need to access private resources securely. The server in this example has an IP address of 192.168.0.172, and the client uses 192.168.0.4. If you installed OpenVPN from an RPM on your distribution media, you need to place the following files in the /etc/openvpn directory. If you installed from the tarball, place the files in the unpacked OpenVPN directory.

First, we must generate a static key that both the client and server can use to encrypt the data travelling between them. We do this with the following command:

```
# openvpn --genkey --secret
static.key
```

If you're using a distribution such as Ubuntu, which doesn't grant access to the root account by default, you can become root by typing `sudo su` and entering the password you log in with.

Inside the static.key file, you'll find the 2,048-bit key used to encrypt all data. This should keep you safe from any attempts to decrypt your communications:

```
#
# 2048 bit OpenVPN static key
#
```

```
-----BEGIN OpenVPN Static key V1-----
1607af9e4462845e4c266dc4d56f5fa4
6f1122d276a52e047cd27818f2fe68f3
...
5ff2648a4434d8726f3aea3d982205c7
e6b2b8735fa3b18e2caf4b9b93fc13ad
-----END OpenVPN Static key V1-----
```

We then need to create a barebones configuration file for the VPN server and place it inside the OpenVPN directory. It should be called openvpn.conf and contain the following:

```
dev tun
ifconfig 10.8.0.1 10.8.0.2
secret static.key
```

This tells OpenVPN to use the virtual TUN network interface device that was created when we installed the software. It also tells the VPN that the local and remote ends of the tunnel are assigned the virtual addresses 10.8.0.1 and 10.8.0.2, and to use the static.key file as the secret encryption key for all communications over the VPN.

Copy the static.key file over to the client computer, place it in the /etc/openvpn directory and create the following client-configuration file (openvpn.conf) in the same directory:

```
remote 192.168.0.172
dev tun
ifconfig 10.8.0.2 10.8.0.1
secret static.key
```

This configuration file in turn tells the client that it is a remote node of the VPN, gives IP addresses to the local and remote ends of the link (which are reversed from the earlier configuration file to reflect the fact that the client is at the opposite end of the link) and specifies that it is to encrypt data using the key stored in static.key.

Now start the server side with the following command:

```
# openvpn --config openvpn.conf --
secret static.key
```

You can put paths into the command-line switches to indicate the locations of `openvpn.conf` and `static.key`. You should then get the following output:

```
Sun Mar 11 15:53:26 2007 OpenVPN
2.0.9 i686-suse-linux [SSL] [EPOLL]
built on Mar 8 2007
Sun Mar 11 15:53:26 2007 IMPORTANT:
OpenVPN's default port number is now
1194, based on an official port number
assignment by IANA. OpenVPN 2.0-
beta16 and earlier used 5000 as the
default port.
Sun Mar 11 15:53:26 2007 TUN/TAP
device tun0 opened
Sun Mar 11 15:53:26 2007 /sbin/
ifconfig tun0 10.8.0.1 pointopoint
10.8.0.2 mtu 1500
Sun Mar 11 15:53:26 2007 UDPv4 link
local (bound): [undef]:1194
Sun Mar 11 15:53:26 2007 UDPv4 link
remote: [undef]
```

Start the client side with the same command line:

```
# openvpn --config openvpn.conf --
secret static.key
Sun Mar 11 16:23:08 2007 OpenVPN
2.0.7 i486-pc-linux-gnu [SSL] [LZO]
[EPOLL] built on Sep 13 2006
Sun Mar 11 16:23:08 2007 IMPORTANT:
OpenVPN's default port number is now
1194, based on an official port number
assignment by IANA. OpenVPN 2.0-
beta16 and earlier used 5000 as the
default port.
Sun Mar 11 16:23:08 2007 TUN/TAP
device tun0 opened
Sun Mar 11 16:23:08 2007 ifconfig tun0
10.8.0.2 pointopoint 10.8.0.1 mtu
1500
Sun Mar 11 16:23:09 2007 UDPv4 link
local (bound): [undef]:1194
Sun Mar 11 16:23:09 2007 UDPv4 link
remote: 192.168.0.172:1194
Sun Mar 11 16:23:19 2007 Peer
Connection Initiated with
192.168.0.172:1194
Sun Mar 11 16:23:20 2007
Initialization Sequence Completed
```

Like other RPM-based servers, OpenVPN also has a startup script in `/etc/init.d`. If you use the

command `/etc/init.d/openvpn start` to start the client side, it will go into daemon mode (run in the background) and will automatically scan `/etc/openvpn` for a file ending in `.conf` to use as its configuration file.

On the client computer, try using the ping command to send some information down the VPN's tunnel:

```
# ping 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes
of data.
64 bytes from 10.8.0.1: icmp_seq=1
ttl=64 time=2.89 ms
64 bytes from 10.8.0.1: icmp_seq=2
ttl=64 time=0.806 ms
...
64 bytes from 10.8.0.1: icmp_seq=8
ttl=64 time=0.807 ms
--- 10.8.0.1 ping statistics ---
8 packets transmitted, 8 received, 0%
packet loss, time 7022ms
rtt min/avg/max/mdev = 0.758/1.058/
2.899/0.696 ms
```

Because we've used the address 10.8.0.1 in the ping command to contact the server, the traffic this command generates needs to travel over the VPN. In fact, on the server, the following two lines should have added themselves to the output of the `openvpn` command in response to the ping:

```
Sun Mar 11 16:34:19 2007 Peer
Connection Initiated with
192.168.0.4:1194
Sun Mar 11 16:34:19 2007
Initialization Sequence Completed
```

If you have a traffic analyser such as Ethereal installed on the server, you should see that you have an extra interface you can monitor called `tun0`. Monitoring the IP address 192.168.0.172 (the server's IP address) will show you nothing but UDP traffic. This is the traffic flowing over port 1194. You can't see any detail. To see more, monitor the `tun0` interface. When tracing problems and monitoring traffic, it's worth remembering that OpenVPN uses UDP for its tunnel by default, rather than TCP.

You can try installing Apache on the server and connecting to it from the client using the URL `http://10.8.0.1`. The default Apache page will appear, tunnelled over the VPN. Ethereal will also pick up the TCP traffic on the TUN interface

10.8.0.1 to and from port 80 as it emerges from the UDP VPN when you connect to the server.

It's worth spending some time setting up a Windows client while you have the point-to-point version of the VPN up and running. Once you know you have an established connection, you can copy over the certificate-based configuration and convert the server to use this more secure configuration. See below for more details.

CERTIFICATE OF CONFIGURATION

It is possible to create certificates and encryption keys for the server and its clients so they can authenticate each other. This also lets us connect more than one client at the same time, rather than just one, as in our point-to-point example above.

OpenVPN has its own method of generating certificates without involving a third-party Certificate Authority, which makes life easier for small organisations and individuals. Bundled scripts control this process, and we'll use them to set up the server first. On the server, enter the OpenVPN directory that was created when you unpacked the source tarball. Open the `easy-rsa` sub-directory and edit the file called `vars`. At the bottom of this file, you'll find entries for country, province, city, organisation and email. Fill in these with your own details and save the file. The `vars` file is actually a script that sets up some useful environment variables containing these values, and we next need to run it as follows:

```
# . ./vars
NOTE: when you run ./clean-all, I
will be doing a rm -rf on /home/jon/
openvpn-2.0.9/easy-rsa/keys
```

Then run the following two scripts in the same directory, as follows:

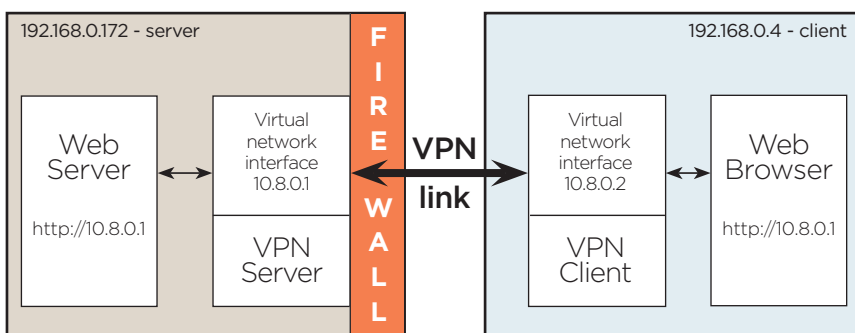
```
# ./clean-all
# ./build-ca
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'ca.key'
----
```

You are about to be asked to enter information that will be incorporated into your certificate request. What you are about to enter is what is called a Distinguished Name or DN. There are quite a few fields but you can leave some blank. For some fields there will be a default value. If you enter '.', the field will be left blank.

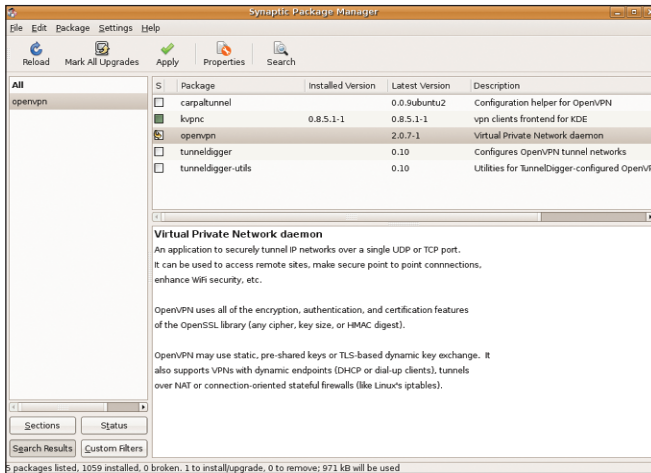
```
-----
Country Name (2 letter code) [UK]:
State or Province Name (full name)
[LONDON]:
Locality Name (eg, city) [LONDON]:
Organization Name (eg, company)
[COMPSHOPPER]:
Organization Unit Name (eg, section)
[]: IT-DEPT
Common Name (eg, your name or your
server's hostname) []: jon
Email Address [root@compshopper.com]:
```

The `clean-all` script clears away any previous output, and `build-ca` creates the certificate

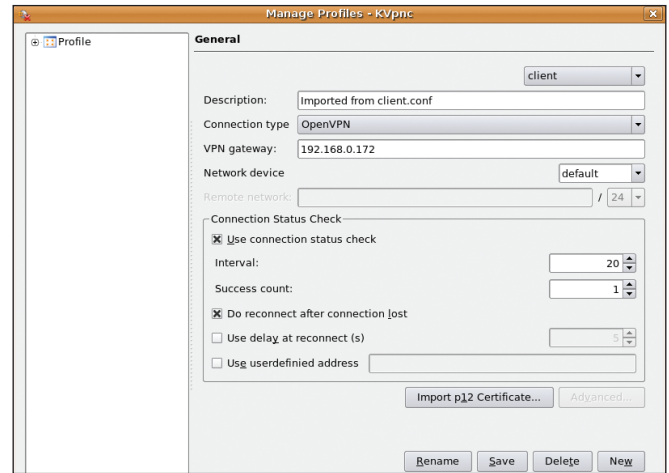
How it works Virtual Private Networks



▲ The client and server connect using virtual interfaces, which are used by programs such as web browsers



▲ Installing OpenVPN with a package manager such as Synaptic on the Ubuntu distribution is very easy



▲ Look at your distribution and you may find VPN utilities such as Kvpnc, which can help configure your server and clients and connect to the server

authority key file, which the client and server will use to verify each other.

Next, we need to generate a private key for the server. It will use this to decrypt incoming traffic that is encrypted by the public key it supplies to the clients when they connect to it. To do this, enter the following:

```
# ./build-key-server server
```

The responses to this script are the same as you entered above, but it also wants a challenge password. Simply press Enter to leave this blank. We tried entering a password and it caused connections to the VPN to fail. The company name that follows it isn't important, either, so you can just press Enter. Answer y to the question about signing the certificate and confirm when asked.

We need to generate private keys and certificates for our clients. We're going to create just one client to show you how, but you need to repeat the procedure for each client and use unique names for them all. There is a slight cheat here: you can use the same certificate and key for each client, but it's potentially not as secure as using individual ones.

To generate the key and certificate for a client, run the following script:

```
# ./build-key client1
```

We've called our client 'client1'. You'll be asked the same questions as above, so give the same answers. This command generates a couple of files, described below.

The final piece of the certificate and key jigsaw is to set up the Diffie-Hellman parameters that the cryptographic algorithms will use. To do so, enter the following command:

```
# ./build-dh
Generating DH parameters, 1024 bit
long safe prime, generator 2
```

This generation process takes a while, because it involves selecting a random prime number with certain properties. Once complete, all the key and certificate information is placed into the keys sub-directory of the easy-rsa directory. Some files are for the server, others are for the client, and some are for both PCs. Here's a table showing where each will live on the network:

FILENAME	DESCRIPTION	REQUIRED BY
Ca.crt	Root CA certificate	Client1 and server
Ca.key	Root CA key	Server
Dh<n>.pem	Diffie Hellman parameters	Server
Server.crt	Server certificate	Server
Server.key	Server key	Server
Client1.crt	Client certificate	Client1
Client1.key	Client key	Client1

Like all good Linux servers, OpenVPN has a configuration file that tells it how to act and where to find these certificates and keys. On the server, the default configuration file is called server.conf and is stored in the sample-config-files subdirectory. You need to edit this file to point to the files just generated, so scroll down to about line 78 and edit the lines beginning ca, cert, key and the entry marked hd. Add the path to the easy-rsa/keys/ subdirectory into the names of the files found there.

We can now start the server part of the VPN as follows:

```
# openvpn --config server.conf
```

Take care to include the path to server.conf if it's not in the current directory. You may see the following error:

```
Options error: Unrecognised option or
missing parameter(s) in server.conf:
245: comp-lzo (2.0.9)
```

If so, you are not alone. Although comp-lzo is listed as a valid option for turning on lzo-based compression, it persistently gave us this error. Our short-term solution was to comment out the offending line with a hash because it doesn't affect the security of the VPN link, but we'd love to hear from any readers who know how to solve the problem properly.

Once running, notice how the startup messages now contain information about the Diffie-Hellman encryption subsystem, routes and so on. We're now in a position to set up and test the client side. To do this, log into the client and change directory to /etc/openvpn. Copy over the .crt, .csr and .key files generated on the server for the client and place them in this directory. Copy over the ca.crt file, too. Then replace the content of the configuration file we used earlier with the following:

```
dev tun
tls-client
ca ca.crt
cert client1.crt
key client1.key
remote 192.168.0.172
pull
user nobody
group nogroup
verb 3
```

The ca, cert and key entries in this file should be fine, as we'll start the client from within this directory using the server startup script, and it'll change to this directory. Test the client with this command to check its output for errors:

```
# openvpn -config openvpn.conf
```

The server should output messages about connections being received, too. You should now be able to issue the command ping 10.8.0.1 to contact the server over the VPN. Try accessing the Apache server you set up earlier via the URL http://10.8.0.1. This should also work. If you now repeat the client side procedure by generating another certificate for the second client, installing the RPM and creating the configuration file, it too should be able to connect.

You can ensure that the server side starts up at system bootup by adding a line to /etc/inittab as follows:

```
vn:35:respawn:openvpn --config <path
to the config file>
```

If your distribution includes the useful insserv command, you can ensure that the VPN starts at boot time by typing this:

```
# insserv /etc/init.d/openvpn
```

This command will also start OpenVPN immediately. You can prevent the server running at boot time again using the -r switch:

```
# insserv -r /etc/init.d/openvpn
```

Next month

SEARCH ENGINE OPTIMISATION
Help search engines index your site